ELSEVIER Linear Algebra and its Applications 338 (2001) 201–218
LINEAR ALGEBRA AND ITS APPLICATIONS
www.elsevier.com/locate/laa

A robust ILU with pivoting based on monitoring the growth of the inverse factors

Matthias Bollhöfer *,[1]

Fakultät II, Institute of Mathematics, Berlin University of Technology, MA 4-5, D-10623 Berlin, Germany

Received 20 July 2000; accepted 22 May 2001

Submitted by V. Mehrmann

Abstract

An incomplete LU decomposition with pivoting is presented that progressively monitors the growth of the inverse factors of L, U. The information on the growth of the inverse factors is used as feedback for dropping entries in L and U. This method often yields a robust and effective preconditioner especially when the system is highly indefinite. Numerical examples demonstrate the effectiveness of this approach. © 2001 Elsevier Science Inc. All rights reserved.

AMS classification: 65F05; 65F10; 65F50

Keywords: Sparse matrices; ILU; Sparse approximate inverse; AINV; Pivoting; Condition estimator

1. Introduction

We consider problems of the form

$$Ax = b, \tag{1}$$

with $A \in \mathbb{R}^{n,n}$ nonsingular and $b \in \mathbb{R}^n$. We focus on problems where A is sparse and where we do not have much information about the system beforehand. These systems might be highly indefinite or ill-conditioned. Since often these systems are very large, solving them is a challenge for numerical algorithms. Sometimes it is exceedingly difficult to solve them by iterative techniques and in these cases direct solvers might be preferred. However, there are situations in which 'general purpose' or

* Tel.: +49-30-314-25741; fax: +49-30-314-79706.
E-mail address: bolle@math.tu-berlin.de; http://www.math.tu-berlin.de/~bolle/. (M. Bollhöfer).

[1] Part of this work was performed while visiting the CERFACS at Toulouse.

PII: S 0 0 2 4 - 3 7 9 5 (0 1) 0 0 3 8 5 - 8

'black-box' iterative solvers are required. The most popular and promising iterative techniques so far are preconditioned Krylov-subspace solvers, see, e.g., [16,19,27]. Among many techniques, preconditioners based on incomplete *LU*-factorizations, see e.g., [21,22,24], are known to give excellent results for many important classes of problems, such as those arising from the discretization of elliptic partial differential equations.

Nevertheless, there are still many situations where incomplete *LU* decompositions give poor results. One often has to play around with the parameters, e.g., to adapt a drop tolerance in the incomplete *LU* decomposition to obtain a successful preconditioner. This is time-consuming since for any problem one has to select the correct values. This reduces the flexibility as a 'black-box' solver. In addition by decreasing parameters to obtain a successful preconditioner we might get enormous fill-in or an unacceptable computational time. In this case direct solvers are the only alternative.

The goal of this paper is to take a closer look at incomplete *LU* decompositions and especially on how entries are dropped. The main key used here for analyzing dropping in the incomplete *LU* decomposition is its strong relation [8,9] to factored sparse approximate inverse methods [2,4,5,20,26]. In an earlier paper [9] comparisons between an incomplete *LU* decompositions with pivoting and a factored approximate inverse with pivoting have shown several examples where the approximate inverse was superior to the *ILU*. Therefore, the rationale of this paper is that the stability of *ILU*s may benefit from exploiting their relationships with approximate inverse techniques.

The main idea is to monitor the growth of the inverse factors of L, U while computing L, U and to use this information as feedback for a refined dropping strategy for the entries of L and U.

2. A simple *ILU* approach

We start with a simple description of a class of incomplete *LU* factorizations. For the solution of (1) we construct an approximate decomposition

$$A \approx LDU,$$

where L, U^{T} are lower triangular matrices with unit diagonal and D is diagonal. One way to construct these decompositions is to partition A as

$$A = \begin{bmatrix} \beta & d \\ c & E \end{bmatrix} \in \mathbb{R}^{n,n}$$

with $\beta \in \mathbb{R}$ and the other blocks have corresponding size. Then A is factored as

$$\begin{bmatrix} \beta & d \\ c & E \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ l & I \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \delta & 0 \\ 0 & S \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 & u \\ 0 & I \end{bmatrix}}_{U}, \tag{2}$$

where

$$S = E - l\delta u \in \mathbb{R}^{n-k,n-k} \tag{3}$$

denotes the so-called Schur-complement. The exact LU-decomposition of A (if it exists) can be obtained by successively applying (2) to the Schur-complement S. Even if there exists a decomposition (2) for A and for S, there is no need to compute l, u, S exactly when constructing a preconditioner. A common approach for reducing fill-in consists of discarding entries in l, u typically of small size and defining the approximate Schur-complement only with these sparsified versions $\tilde{l}, \tilde{u}$ of l, u. Here we will concentrate on

$$\tilde{S} = E - \tilde{l}d - (c - \tilde{l}\beta)\tilde{u} \tag{4}$$

as one possible definition of an approximate Schur-complement. This kind of approximate Schur-complement is used in e.g. [9,28]. The main reason to use this kind of approximate Schur-complement is the following observation. Eq. (4) can be obtained from the lower right block of $\tilde{L}^{-1}A\tilde{U}^{-1}$. Since it is our goal to monitor the growth of the inverse factors L^{-1}, U^{-1} later on, (4) is a reasonable choice.

At a few places in the algorithms we use a MATLAB-like notation [1] for convenience. For a column vector $p \in \mathbb{R}^m$ and a matrix $M \in \mathbb{R}^{m,n}$ we define for numbers $1 \leqslant j \leqslant k \leqslant m$ and $1 \leqslant l \leqslant n$,

$$p(j:k) = \begin{pmatrix} p_j \\ p_{j+1} \\ \vdots \\ p_k \end{pmatrix}, \quad M(j:k,l) = \begin{pmatrix} a_{j,l} \\ a_{j+1,l} \\ \vdots \\ a_{k,l} \end{pmatrix}.$$

for row vectors and rows $M(l, j:k)$ an analogous definition is used. The associated ILU algorithm is roughly as follows.

Algorithm 1 (*Incomplete LU factorization* (*ILU*)). Given $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$ and drop tolerance $\tau \in [0, 1]$. Compute $A \approx LDU$.

Set $L = U = D = I,\ S = A$.
for $i = 1, \ldots, n-1$
  if desired, include a pivoting step.
  $d_{ii} = s_{ii}$
  $c = S(i+1:n, i),\ d = S(i, i+1:n)$
  $p = c/d_{ii},\ q = d/d_{ii}$
  Drop those components of $|p|, |q|$ which are less than τ.
  $L(i+1:n, i) = p,\ U(i, i+1:n) = q$
  Denote by $\hat{S}$ the submatrix $\hat{S} = (s_{kl})_{k,l=i+1,\ldots,n}$ of S.
  Replace $\hat{S}$ by $\hat{S} = \hat{S} - pd - (c - p\beta)q$
end
$d_{nn} = s_{nn}$

Practical versions of incomplete LU decompositions are typically implemented in a slightly different way. It is usually not advisable to update the whole $\hat{S}$ by a rank-1 or rank-2 modification. Instead, typically only the leading row of $\hat{S}$ is computed,

and the transformations on the other rows are postponed. This corresponds to the so-called I, K, J version of Gaussian elimination [26]. Besides saving memory, this approach is easier to implement since all updates and modifications are performed only once for each row. Thus one can use simple sparse row storage schemes, e.g. the compressed sparse row (CSR) format. For details see [26].

Algorithm 2 (*Incomplete LU factorization, I, K, J version*). Given $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$ and a drop tolerance $\tau \in [0, 1]$. Compute $A \approx LDU$.

$L = U = D = I,\ S = A.$
for $i = 1, \ldots, n$
  $w = (a_{i,1}, \ldots, a_{i,n})$
  for $k = 1, \ldots, i-1$
    $w_k = w_k / d_{kk}$
    if $|w_k| \leqslant \tau$, $w_k = 0$, **else** $w(k+1:n) = w(k+1:n) - w_k U(k, k+1:n)$
  end
  $d_{ii} = w_i$
  $L(i, 1:i-1) = w(1:i-1)$,
  for all $j > i$: $u_{ij} = w_j / w_i$. **if** $|u_{ij}| \leqslant \tau$, $u_{ij} = 0$
end

Mathematically Algorithm 2 can be read as a special version of Algorithm 1, if the approximate Schur-complement is replaced by

$$\hat{S} = \hat{S} - p\beta q.$$

Clearly this replacement would also end up in an exact LU decomposition once we do not drop entries anymore.

Both the algorithms, Algorithms 1 and 2, represent a good compromise in many numerical examples. However, there are problems like highly indefinite problems which are extremely sensitive with respect to dropping and only using very small drop tolerances τ might lead to successful preconditioners. Our goal is to develop new strategies of dropping and pivoting for these kinds of matrices to obtain a robust preconditioner that is less sensitive to errors invoked by dropping entries.

3. Stabilized ILU

One problem in dropping entries in Algorithms 1 or 2 is that we do not have control of the changes which are affected by dropping. One way to get a more reliable dropping criterion is to take the norm of the ith row of A into account, e.g. replace τ by $\tau \cdot \|e_i^{\mathrm{T}} A\|_1$. This is essentially what the ILUT-Algorithm [24] does. Often such a strategy is a very good compromise but clearly there may still be cases where we could end up in a poor preconditioner.

Algorithms 1 and 2 can be supplemented with pivoting. When column pivoting is added to Algorithm 2 it essentially corresponds to the ILUTP-Algorithm which is part of SPARSKIT, see e.g. [25,26]. So far we have ignored this option to have a clearer presentation. Later on, we will return to this point and finally include pivoting. For simplicity let us consider the algorithms without pivoting at this stage.

Scaling τ as well as using pivoting still cannot control how dropping small entries effects the quality of the preconditioner. The main problem is that the preconditioned system is of type $L^{-1}AU^{-1}D^{-1}$. Whenever we drop elements in L, U that are somehow "small", we do not know the impact of these changes with respect to the preconditioned system $L^{-1}AU^{-1}D^{-1}$ since L^{-1} and U^{-1} are not available. It seems that a natural way to get more control on dropping is to include information on the approximate inverse $W^{\mathrm{T}} \approx L^{-1}$ and $Z \approx U^{-1}$ into the algorithm. Recently it has been shown in [8] that Algorithm 1 has a strong relation to sparse approximate inverse preconditioners. Without going into the details, we will roughly describe the idea of AINV-type algorithms [2,4,5,9]. The idea is to directly comput the upper triangular matrices W, Z such that $W^{\mathrm{T}}AZ = D$, with a diagonal matrix D. The version which we will focus on is the so-called right looking AINV, where W and Z are updated by a rank-1 update. Essentially a biorthogonalization process for W and Z is performed, in which $W^{\mathrm{T}}A$ and $Z^{\mathrm{T}}A^{\mathrm{T}}$ are transformed step by step to upper triangular form. Clearly this only holds if no dropping is applied to W, Z.

Algorithm 3 (*Factored approximate INVerse, rank-*1 *update version*). Given $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$ and a drop tolerance $\tau \in [0, 1]$. Compute $A^{-1} \approx ZD^{-1}W^{\mathrm{T}}$.

$p = q = (0, \ldots, 0) \in \mathbb{R}^n,\ Z = W = I_n.$
for $i = 1, \ldots, n$
  $p_i = e_i^{\mathrm{T}}AZe_i, q_i = e_i^{\mathrm{T}}W^{\mathrm{T}}Ae_i$
  for $j = i + 1, \ldots, n$
    $p_j = e_j^{\mathrm{T}}AZe_i/pi,\ \ q_j = e_i^{\mathrm{T}}W^{\mathrm{T}}Ae_j/qi$
    $W(1 : i, j) = W(1 : i, j) - W(1 : i, i)p_j,$
    $Z(1 : i, j) = Z(1 : i, j) - Z(1 : i, i)q_j$
  end
  for all $k \leqslant i, l > i$: drop w_{kl}, if $|w_{kl}| \leqslant \tau$ and drop z_{kl}, if $|z_{kl}| \leqslant \tau$
  $d_{ii} = p_i.$
end

In principle we could modify Algorithm 1 such that the inverses of its triangular factors L, U are computed on the fly. For this purpose we supplement Algorithm 1 with a progressive inversion of L, U. At step $i - 1$, U is of the form

$$U = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ O & 1 & O \\ O & O & I \end{bmatrix}$$

and the ith step will compute the entries U_{23} and add them to the current U to get U_{new}. Let q^{T} be the row vector $q^{\mathrm{T}} = e_i^{\mathrm{T}}(U - I)$. Note that the 'diagonal' element q_i of q is zero. Then

$$U_{\text{new}} = U + e_i q^{\mathrm{T}} = (I + e_i q^{\mathrm{T}})U.$$

It follows that

$$U_{\text{new}}^{-1} = U^{-1}(I + e_i q^{\mathrm{T}})^{-1} = U^{-1}(I - e_i q^{\mathrm{T}}).$$

Of course, analogous arguments hold for L. This provides a formula for progressively computing $L^{-\mathrm{T}}, U^{-1}$ throughout the algorithm. We call the inverse factors Z, W as in Algorithm 3. With these additional factors Z, W and a modified Schur-complement it was shown in [8] that the supplemented version of Algorithm 1 is essentially equilvalent to Algorithm 3.

Theorem 1. *Suppose that Algorithm* 1 *is supplemented with a progressive inversion of L,U (before p, q are sparsified). Suppose in addition that in step i of Algorithm* 1 *an entry l_{ji} is discarded only if $|l_{ji}| \cdot \max(\{1\} \cup \{|w_{ki}| : k = 1, \ldots, i-1\}) \leqslant \tau$, $i = 1, \ldots, n$. Suppose that in Algorithms* 1 *and* 3 *for $k \leqslant i$ and $l > i$, w_{kl} is dropped if $|w_{kl}| \leqslant \tau$. If the (modified) Schur-complement $\hat{S} = (s_{kl})_{k,l>i}$ is defined as the lower $(n-i) \times (n-i)$ block of $W^{\mathrm{T}}A$, then we have for any $k > l$:*

$$\left|(I - WL^{\mathrm{T}})_{lk}\right| \leqslant 2\tau(k - l)$$

and the diagonal entries of D are those of p. Here $(I - WL^{\mathrm{T}})_{lk}$ denotes the entry of $I - WL^{\mathrm{T}}$ at position $(l,\ k)$.

Proof. See [8]. □

The most interesting point about this relation is that Theorem 1 requires to modify the dropping strategy for L (and similarly for U). Now typically applying dropping to sparse approximate inverse factors is less harmful than for incomplete LU decompositions, because in dropping small entries of size τ in W, Z the effective error in $W^{\mathrm{T}}AZ$ is only between linear and quadratic with respect to τ. And $W^{\mathrm{T}}AZ$ is the matrix which needs to be transformed to an approximately diagonal matrix D. On the other hand if we apply dropping to the factors L, U of an *ILU* the related effect is rational since we do not know in advance the effect for $L^{-1}AU^{-1}$. But for preconditioning, this is precisely what we need to know. So if we can construct an *ILU* that is somehow almost equivalent to an approximate inverse, then we might hope that dropping is more reliable and the resulting preconditoner is much more efficient for those situations where dropping has a serious impact on the quality of the preconditioner. Numerical results in [9] illustrate that for some extremely indefinite and ill-conditioned problems the approximate inverse behaves better than an *ILU*.

To turn the result of Theorem 1 into an algorithm we will certainly not invert L, U in Algorithm 1. Let us take a look at the criterion for dropping entries in L. We need to know $\max(\{1\} \cup \{|w_{ki}| : k = 1, \ldots, i-1\})$, which means we need to know

the ith row of L^{-1}, i.e., $w_{ji} \approx (L^{-1})_{ij}$ for all $j < i$. At least it would be convenient to have an estimate for $\|e_i^{\mathrm{T}} L^{-1}\|_1$ which could serve as a substitute for $\{|w_{ki}| : k = 1, \ldots, i-1\}$. Analogously an estimate for $\|U^{-1} e_i\|_1$ needs to be computed to apply a similar dropping strategy to the entries of U.

Algorithm 4 (*Scheme of an ILU with estimates for the growth of* $\|L^{-1}\|, \|U^{-1}\|$). Given $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$, a drop tolerance $\tau \in [0, 1]$. Compute $A \approx LDU$.

Set $L = U = D = I,\ S = A$.
for $i = 1, \ldots, n-1$
  if desired, include a pivoting step.
  $d_{ii} = s_{ii}$
  $c = S(i+1:n, i),\ d = S(i, i+1:n)$
  $p = c/d_{ii},\ q = d/d_{ii}$
  Compute estimates for $\eta_L = \|e_i^{\mathrm{T}} L^{-1}\|_1,\ \eta_U = \|U^{-1} e_i\|_1$
  Drop those components of $|p|$ which are less than $\tau/\max\{1, \eta_L\}$.
  Drop those components of $|q|$ which are less than $\tau/\max\{1, \eta_U\}$.
  $L(i+1:n, i) = p,\ U(i, i+1:n) = q$
  Denote by $\hat{S}$ the submatrix $\hat{S} = (s_{kl})_{k,l=i+1,\ldots,n}$ of S.
  Replace $\hat{S}$ by $\hat{S} = \hat{S} - pd - (c - p\beta)q$
end
$d_{nn} = s_{nn}$

In order to get a helpful estimate for $\eta_L = \|e_i^{\mathrm{T}} L^{-1}\|_1, \eta_U = \|U^{-1} e_i\|_1$ we use a general condition estimator for upper triangular matrices from [10,18]. Here we consider an adapted version for lower triangular matrices. The condition estimator is based on solving a system with a lower triangular matrix L where the right-hand side b only consists of ± 1 and the signs are chosen to successively maximize the solution x of $Lx = b$. Another look at this condition estimator shows that absolute values $|x_i| = \|e_i^{\mathrm{T}} L^{-1} b\|_1 / \|b\|_1$ precisely estimate $\|e_i^{\mathrm{T}} L^{-1}\|_1$. To apply this estimator to our problem we will consider $Lx_L = b_L$ and $U^{\mathrm{T}} x_U = b_U$ to get estimates for $\eta_L = \|e_i^{\mathrm{T}} L^{-1}\|_1$ and $\eta_U = \|U^{-1} e_i\|_1$. The idea of the following algorithm, Algorithm 5, is to decide in each step i of the forward substitution process, whether the ith component b_i of b should be set 1 or to -1 in order to maximize $|x_i|$. The strategy in Algorithm 5 takes an additional look-ahead on the growth of succeeding components x_j, $j > i$.

Algorithm 5 (*Condition estimator for* (L^{-1})). Let $L = (L_{ij})_{ij} \in \mathbb{R}^{n,n}$ be unit lower triangular. Compute $Lx = b$, where $b^{\mathrm{T}} \in (\pm 1, \ldots, \pm 1)$.

Let $v = x = (0, \ldots, 0)^{\mathrm{T}} \in \mathbb{R}^n$
for $i = 1, \ldots, n$
  $\mu^+ = 1 - v_i,\ \ \mu^- = -1 - v_i$

$v^+ = v(i+1:n) + L(i+1:n,i)\mu^+,\ \ v^- = v(i+1:n) + L(i+1:n,i)\mu^-$
if $|\mu^+| + \|v^+\|_1 > |\mu^-| + \|v^-\|_1 :\ \ x_i = \mu^+,\ v(i+1:n) = v^+$
else $x_i = \mu^-,\ v(i+1:n) = v^-$
end

The condition estimator from Algorithm 5 was originally developed for full matrices. One can verify that it can be easily adapted to the sparse case by using copies v^+, v^- which are updated in each step. We skip these details. In principle one could also use different condition estimators, e.g. [6,7]. What we really need is not an estimate for the norm of L^{-1} but an estimate for the norm of each row of L^{-1}. From this point of view to take as right-hand side a vector y which only consists of ± 1 is reasonable and attractive for this problem.

The computation of x in Algorithm 5 can be interlaced with Algorithm 4, i.e. during the *ILU* factorization, estimates for $\eta_L = \|e_i^{\mathrm{T}} L^{-1}\|_1$ and $\eta_U = \|U^{-1} e_i\|_1$ can be computed simultaneously. This is demonstrated in Algorithm 6.

Algorithm 6 (*ILU factorization with estimates of* $\|e_i^{\mathrm{T}} L^{-1}\|,\ \|U^{-1} e_i\|$). Given $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$, a drop tolerance $\tau \in [0,1]$. Compute $A \approx LDU$.

Set $L = U = D = I,\ S = A$.
Let $v = w = x = y = (0,\ldots,0)^{\mathrm{T}} \in \mathbb{R}^n$
for $i = 1,\ldots,n-1$
  if desired, include a pivoting step.
  $d_{ii} = s_{ii}$
  $c = S(i+1:n,i),\ d = S(i,i+1:n)$
  $p = c/d_{ii},\ q = d/d_{ii}$
  Compute estimate for $\eta_L \approx \|e_i^{\mathrm{T}} L^{-1}\|_1$:
    $\mu^+ = 1 - v_i,\ \ \mu^- = -1 - v_i$
    $v^+ = v(i+1:n) + L(i+1:n,i)\mu^+,\ \ v^- = v(i+1:n) + L(i+1:n,i)\mu^-$
    if $|\mu^+| + \|v^+\|_1 > |\mu^-| + \|v^-\|_1 :\ x_i = \mu^+,\ \ v(i+1:n) = v^+$
    else $x_i = \mu^-,\ v(i+1:n) = v^-$
    Compute estimate for $\eta_U \approx \|U^{-1} e_i\|_i$:
    $v^+ = 1 - w_i,\ \ v^- = -1 - w_i$
    $w^+ = w(i+1:n) + U(i,i+1:n)^{\mathrm{T}} v^+,\ \ w^- = w(i+1:n) + U(i,i+1:n)^{\mathrm{T}} v^-$
    if $|v^+| + \|w^+\|_1 > |v^-| + \|w^-\|_1 :\ y_i = v^+, w(i+1:n) = w^+$
    else $y_i = v^-, w(i+1:n) = w^-$
  $\eta_L = |x_i|, \eta_U = |y_i|$
  Drop those components of $|p|$ which are less than $\tau/\max\{1, \eta_L\}$.
  Drop those components of $|q|$ which are less than $\tau/\max\{1, \eta_U\}$.
  $L(i+1:n,i) = p,\ \ U(i,i+1:n) = q$
  Denote by $\hat{S}$ the submatrix $\hat{S} = (s_{kl})_{k,l=i+1,\ldots,n}$ of S.
  Replace $\hat{S}$ by $\hat{S} = \hat{S} - pd - (c - p\beta)q$
end
$d_{nn} = s_{nn}$

Algorithm 6 as it stands now describes the main changes compared with standard *ILU* techniques.

Next we introduce pivoting. Apart from some problems that are known not to necessitate permutations, in general we have to include pivoting to guarantee stability for the construction of the decomposition itself. Even if we would not divide by zero, small entries have similar effect. Even worse, in our construction the threshold for dropping is coupled with the growth of the norm of the inverse factors L^{-1}, U^{-1}. Small diagonal entries in absolute value would immediately result in a huge norm of the inverse. In other words, without pivoting there is a potential danger that the thresholds $\tau/\max\{1, \eta_L\}$, $\tau/\max\{1, \eta_U\}$ are too small to drop anything. This argument illustrates that pivoting and dropping go hand in hand and that the adaptive threshold only makes sense if it is safeguarded by a pivoting process that controls the growth of the diagonal pivots. From this point of view it pays off to include pivoting in order to be able to drop more entries. We define permutation vectors π, σ such that $A(\pi, \sigma) = LD(\pi, \sigma)U$ provided that no dropping is applied. In principle, applying permutation matrices Π, Σ to (2), changes this equation to

$$\begin{pmatrix} 1 & O \\ O & \Pi^{\mathrm{T}} \end{pmatrix} \begin{bmatrix} \beta & d \\ c & E \end{bmatrix} \begin{pmatrix} 1 & O \\ O & \Sigma \end{pmatrix} = \begin{bmatrix} 1 & O \\ \Pi^{\mathrm{T}} l & I \end{bmatrix} \begin{bmatrix} \delta & 0 \\ 0 & \Pi^{\mathrm{T}} S \Sigma \end{bmatrix} \begin{bmatrix} 1 & u\Sigma \\ O & I \end{bmatrix}.$$

This illustrates how S, L and U have to be adapted. It should also be obvious that in practice one will not explicitly interchange rows of L and columns of U but instead one uses index vectors.

In principle we can introduce a pivoting process to Algorithm 6 which ensures that in the permuted matrix $|s_{ii}| \geqslant \alpha \max_{j \geqslant i} |s_{ji}|$ and $|s_{ii}| \geqslant \alpha \max_{j \geqslant i} |s_{ij}|$. This guarantees that after the division by s_{ii} the entries of p, q are less than $1/\alpha$ in absolute value. Here the parameter $\alpha \in [0, 1]$ is chosen a priori. The choice $\alpha = 1$ refers to strict pivoting, i.e. the maximum entry in absolute value will become s_{ii} while any smaller choice of α causes only pivoting if the diagonal entry is much smaller than the maximum entry of $|S(i + 1 : n, i)|$ (respectively, $|S(i, i + 1 : n)|$). Now we can go one step further and use the freedom in the choice of pivots to add a strategy of Markovitz type [12]. This strategy is frequently used in direct methods [11,15]. The idea is to balance the stability of the pivots and the preservation of the sparsity. In this context this means that in column i of S we consider the set of row pivots k such that $|s_{ki}| \geqslant \alpha \max_{j \geqslant i} |s_{ji}|$ and among these we take the one with the minimum number of nonzeros in $S(i : n, k)$. The same process needs to be repeated for row i of S. This leads to an alternating scheme of row and column pivoting. Since we do not want this procedure to become an infinite loop or that the final diagonal entry falls below a certain level we keep track on $\max_{j \geqslant i} |s_{ji}|$ and $\max_{j \geqslant i} |s_{ij}|$. A local pivoting step can then look as follows.

Algorithm 7 (*Local pivoting with respect to fill-in*). Given $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$ and a pivoting tolerance $\alpha \in [0, 1]$.
Let $S(i : n, i : n)$ denote the Schur-complement on entry to step i of Algorithm 6.

$z = 0$
while pivots not satisfactory
  $z = \max\{z, \max_{j \geqslant i} |s_{ij}|\}$
  for all l with $|s_{il}| \geqslant \alpha z$ choose l s.t. $S(l, i:n)$ has minimum number of nonzeros
  Interchange $S(i:n, i) \leftrightarrow S(i:n, l)$, $U(1:i-1, i) \leftrightarrow U(1:i-1, l)$
  $z = \max\{z, \max_{j \geqslant i} |s_{ji}|\}$
  for all k with $|s_{ki}| \geqslant \alpha z$ choose k s.t. $S(i:n, k)$ has minimum number of nonzeros
  Interchange $S(i, i:n) \leftrightarrow S(k, i:n)$, $L(i, 1:i-1) \leftrightarrow L(k, 1:i-1)$
end

The while loop only terminates if no more interchanges are performed.

It is clear that we can insert Algorithm 7 on entry to step i of Algorithm 6. The only thing we have to do is to store the column and row permutations in additional permutation vectors, say π and σ and to interchange the components of w, y and v, x analogously. The changes in w, y correspond to the interchanges of column i and l of U and v, x are changed in the same way as the rows i, k of L. We omit the changes of Algorithm 6, since they are quite obvious.

A slight final change in Algorithm 6 consists of locally adapting τ to the size Ae_i and $S(i:n, i)$. As mentioned previously, Algorithm 2 with τ replaced by $\tau \cdot \|e_i^{\mathrm{T}} A\|_1$ essentially corresponds to the ILUT-Algorithm [24]. As analogy here we replace $\tau/\max\{1, \eta_L\}$ to

$$\tau/\max\{1, \eta_L\} \to \tau \min\{\|Ae_i\|_1, \|S(i:n, i)\|_1\}/\max\{1, \eta_L\} \tag{5}$$

and

$$\tau/\max\{1, \eta_U\} \to \tau \min\left\{\left\|e_i^{\mathrm{T}} A\right\|_1, \|S(i, i:n)\|_1\right\}\Big/\max\{1, \eta_U\}. \tag{6}$$

If the information on the Schur-complement is available, it makes sense to include the norm of the ith row/column of the Schur-complement as well. Especially when the corresponding row of the Schur-complement has significantly smaller entries we could encounter $\|S(i:n, i)\|_1 \ll \|Ae_i\|_1$ or $\|S(i, i:n)\|_1 \ll \|e_i^{\mathrm{T}} A\|_1$.

3.1. The ILUSTAB-Algorithm

We have now completed the changes of Algorithm 6. In the following section the name ILUSTAB will refer to Algorithm 6 including the pivoting step from Algorithm 7 and the modified thresholds (5) and (6).

There are two major differences between Algorithm 1 and ILUSTAB. These are the inclusion of a condition estimator which directly controls the dropping tolerance τ and the application of pivoting. The condition estimation is motivated by the strong relations between incomplete LU factorizations and factored approximate inverse preconditioners. The use of pivoting here is important to control the growth norms of $\|L^{-1}\|$, $\|U^{-1}\|$.

4. Numerical results

This section presents numerical experiments to validate the algorithms. So far, the ILUSTAB-Algorithm is implemented in MATLAB [1]. We will refer to it as ILUSTAB.

- The matrices are initially reordered using the symmetric minimum degree ordering [17]. Since the matrices tested here are unsymmetric, this reordering does not guarantee that the fill-in will be less.
- An a priori scaling is used, such any row of the given matrix has unit 1-norm. Like the symmetric reordering, scaling does not necessarily simplify the problem.
- For the pivoting process $\alpha = 0.1$ is used.
- Different values were used for the drop tolerance $\tau = 0.1, 0.3$.

For the numerical experiments several unsymmetric matrices from the Harwell–Boeing Collection [13,14,23] were chosen.

The results are compared with

- LU from **MATLAB** also with pivoting tolerance $\alpha = 0.1$.
- LUINC from **MATLAB** with $\alpha = 0.1$ and drop tolerances $\tau = 0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}$.
- ILUTP from SPARSKIT using the same tolerance $\alpha = 0.1$ for pivoting but $\tau = 0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}$ for dropping.
- Since Algorithm 6 is located between incomplete *LU* decompositions and factored sparse approximate inverse techniques, additional results for a factored sparse approximate inverse with pivoting (AINVP) from [9] are added. For details of this algorithm we refer to [9] and earlier papers [3–5]. In this algorithm analogous parameters $\alpha = 0.1$ and $\tau = 0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}$ were tested.

The numerical results for ILUTP [26] and AINVP [9] were performed on an SGI workstation with two 190 MHz R10000 (IP25) processors under IRIX 6.2 and 512 MB memory.

As iterative solvers GMRES(30) [27] is used. The iteration was stopped after the residual norm was less than $\sqrt{\text{eps}}$ times the initial residual norm, where eps $\approx 2.2204 \times 10^{-16}$ denotes the machine precision. The iteration was stopped after 500 steps. Every iterative solution which broke down or did not converge within the number of steps was noted as a failure.

We briefly describe the results for several matrices and then give detailed numerical results for several selected examples.

To give a rough idea on how the method performed on the Harwell–Boeing collection we simply summarize in Table 1 which method successfully solved how many problems with respect to the drop tolerance τ. The free spaces in Table 1 mean that the algorithms were not tested with these tolerances. The tests were done on 94 matrices from the Harwell–Boeing collection.

Note that there were only two matrices which could not be solved with ILUSTAB for $\tau \in \{0.3, 0.1\}$. These are the matrices *facsimile/fs7603*, *grenoble/gre216b*. These matrices could be solved with $\tau = 0.01$. Note that LUINC could also not

Table 1
Summary of results—successful computation; Harwell–Boeing collection (94 test matrices)

Preconditioner	Drop tolerance τ					
	0.3	0.1	0.01	10^{-3}	10^{-4}	10^{-5}
ILUSTAB	89	92	94			
LUINC		31	52	68	79	87
ILUTP		53	69	78	84	90
AINVP		57	78	88	90	91

solve *facsimile/fs7603* and for *facsimile/fs7603* ILUTP needed $\tau = 10^{-4}$. For *grenoble/gre216b* LUINC and ILUTP needed $\tau = 10^{-5}$.

We now comment on several matrices from the Harwell–Boeing-collection. This collection consists of many matrices from different areas. Related matrices are put together in a group and comments are done with respect to these groups. For some selected examples we will show separate tables. In each table (e.g., Table 2) we will present the choice of the drop tolerance τ and the related fill-in factor (that is the ratio of the number of nonzeros of $L + U$ divided by the number of nonzeros of A). Next the number of iteration steps using GMRES(30) is shown. For the **MATLAB** algorithms LU, ILUSTAB and LUINC we use the flop count as measure for the number of operations. The flop count is split into the flops required for the decomposition and the flops to solve a linear system using GMRES(30).

- CHEMWEST: These matrices are some of those for which LUINC and ILUTP needed smallest drop tolerances to be successful while ILUSTAB was able to solve all of them already for $\tau = 0.3$. AINVP could solve most of these matrices for $\tau = 0.01$. For *west0655*, *west1505* and *west2021*, $\tau = 10^{-3}$ was sufficient.
  Detailed results for the biggest WEST-matrix are given in Table 2. The results for the other matrices are similar.
- FACSIMILE: LUINC from **MATLAB** could not solve most of these matrices for $\tau = 0.1, 0.01$. For $\tau = 10^{-3}$ it was able to solve 50% of them and for $\tau = 10^{-4}, 10^{-5}$ only *fs1836*, *fs7602*, *fs7603* could not be solved. For those problems that could be solved, the fill-in was moderate and the number of iteration steps was small.
  In contrast to this ILUSTAB could solve all of these matrices already for $\tau = 0.3$ except *fs7603* which could not be solved. The fill-in was small as well. The number of iteration steps was small except for *fs7602* which required 60 steps for $\tau = 0.3$ and 31 for $\tau = 0.1$.
  ILUTP solved most of these problems for $\tau = 0.1$. All problems including *fs7603* were solved for $\tau = 10^{-4}, 10^{-5}$.
  For those problems that could be solved the fill-in was small. The largest number of iterations were 155 for *fs7602* and $\tau = 0.1$, 62 for *fs7602* and $\tau = 10^{-3}$. For all other methods it was less, if they could be solved at all.

Table 2
Matrix CHEMWEST/WEST2021

Method	τ	$\frac{nnz(L+U)}{nnz(A)}$	No. iteration steps	Flops	
				Decomposition	Solve
Sparse LU		5.6	1	1.2×10^6	2.7×10^5
ILUSTAB	0.3	1.6	20	6.8×10^5	2.9×10^6
	0.1	1.7	14	6.7×10^5	1.7×10^6
LUINC	10^{-1}	0.7	–	2.2×10^4	–
	10^{-2}	0.9	–	3.0×10^4	–
	10^{-3}	1.2	–	4.6×10^4	–
	10^{-4}	1.6	–	8.7×10^4	–
	10^{-5}	1.9	6	1.2×10^5	5.5×10^5
ILUTP	10^{-1}	1.0	–		
	10^{-2}	1.4	–		
	10^{-3}	1.9	–		
	10^{-4}	2.5	–		
	10^{-5}	3.1	14		
AINVP	10^{-1}	2.5	–		
	10^{-2}	6.4	–		
	10^{-3}	9.9	30		

AINVP needed $\tau = 0.1$ to solve most of these sample matrices, except *fs5414, fs7602*, and *fs7603*. GMRES(30) only needed a few number of steps to compute the solution. The remaining problems were solved for $\tau = 10^{-2}, 10^{-4}$ and 10^{-5}. For $I - A$ the numerical results are much better.

- GEMAT: ILUSTAB could not solve these matrices for $\tau = 0.3$ but for $\tau = 0.1$. LUINC could solve these matrices for $\tau = 10^{-4}$ but with roughly four times of the fill-in of ILUSTAB.
  ILUTP could solve these matrices for $\tau = 10^{-3}$ but with more than twice as much fill-in as ILUSTAB. For these matrices the *LU* decomposition needed more than 70 times of fill than the initial matrix.
  For *gemat12* see Table 3. The results for *gemat11* are similar.
- GRENOBLE: for $\tau = 0.1$ LUINC could only solve *gre115*, *gre216a*, *gre343*, *gre512*. But even for some of those the fill-in factor was already enormous (e.g. 5.9 for *gre216a*, 7.7 for *gre343*, 11.3 for *gre512*). The same problem occurred for the other matrices that could only be solved for smaller τ. All matrices could finally be solved with $\tau = 10^{-5}$.
  ILUSTAB solved all matrices except *gre216b*, *gre1107*, for $\tau = 0.3$. The fill-in was slightly better (e.g. i.e. 3.8 for *gre216a*, 4.9 for *gre343*, 7.8 for *gre512*). *gre 1107* could be solved with $\tau = 0.1$ but with a full-in factor 7.4. This was still better than LUINC, which needed $\tau = 10^{-3}$ and produced a fill-in factor 23.0!

Table 3
Matrix GEMAT/GEMAT12

Method	τ	$\frac{nnz(L+U)}{nnz(A)}$	No. iteration steps	Flops	
				Decomposition	Solve
Sparse LU		73.5	1	1.7×10^9	1.0×10^7
ILUSTAB	0.3	1.0	–	1.4×10^6	–
	0.1	1.3	67	2.0×10^6	3.5×10^7
LUINC	10^{-1}	0.6	–	1.8×10^5	–
	10^{-2}	1.4	–	1.6×10^5	–
	10^{-3}	2.7	–	1.3×10^7	–
	10^{-4}	5.2	10	5.4×10^7	5.7×10^6
	10^{-5}	9.2	5	1.3×10^8	3.9×10^6
ILUTP	10^{-1}	1.0	–		
	10^{-2}	2.0	–		
	10^{-3}	3.4	17		
	10^{-4}	5.2	7		
	10^{-5}	7.4	4		
AINVP	10^{-1}	6.1	–		
	10^{-2}	17.5	–		
	10^{-3}	22.7	22		

For $\tau = 0.1$, ILUTP could solve *gre115*, *gre185*, *gre216a*. But even then the fill-in factor was sometimes large (i.e. 7.0 for *gre216a*, 12.6 for *gre512*). The same problem occurred for the other matrices that could only be solved for smaller τ. For example *gre1107* could be solved with $\tau = 10^{-3}$ and fill-in factor 21.3. All matrices could finally be solved with $\tau = 10^{-5}$.
The problem with the fill-in also extremely affects the sparse *LU* decomposition. For example *gre1107* required a fill-in factor 44.1!
For those problems that could be solved by one of these methods the number of iteration steps was moderate.
AINVP could solve all matrices except *gre216b*, *gre1107*, for $\tau = 10^{-2}$. It needed a small number of iteration steps, but the fill-in was sometimes giant.
We like to note that the matrix which is stored in the collection does not really reflect the underlying application. In fact one should use $I - A$ instead of A.

- LNS: ILUSTAB solved them all for $\tau = 0.3$. The fill-in was moderate (3.6 for *lns3937* was already maximum) and so was the number of iteration steps (at most 29).
LUINC could not solve any of these matrices for $\tau = 0.1, 0.01$ but *lns511*, *lnsp511*, *lns3937*, *lnsp3937* for $\tau = 10^{-3}$. The biggest matrices required twice as much fill-in as ILUSTAB.

ILUTP could solve the two smallest matrices for $\tau = 0.1$ and the medium size matrices for $\tau = 10^{-3}$.
The two biggest matrices could only be solved for $\tau = 10^{-5}$.
The small matrices were solved by AINVP using $\tau = 0.1$. For the bigger matrices $\tau = 10^{-3}$ was necessary and the fill-in also increased drastically (compared with the small matrices). For *lns3937* see Table 4. The results for *lnsp3937* were quite similar. Surprisingly for $\tau = 0.1$ the fill-in was worst. The reason might be that the drop tolerance was too rough and the computed factors were overlayed by the approximation errors.

- NUCL: ILUSTAB could solve all matrices for $\tau = 0.3$ but the fill-in was poor, e.g., 28.6 for *nnc1374*. The number of iteraton steps was at most 28.
LUINC did not solve any of these matrices for $\tau = 10^{-1}, \ldots, 10^{-5}$.
ILUTP could solve all the problem for $\tau = 10^{-3}$ and a better fill-in factor than ILUSTAB (e.g. 6.6 for *nnc1374* but 463 iteration steps).
Except *nnc666*, which could be solved using $\tau = 10^{-4}$, AINVP was not able to solve any of these matrices for $\tau = 10^{-1}, \ldots, 10^{-5}$.

Table 4
Matrix LNS/LNS3937

Method	τ	$\frac{nnz(L+U)}{nnz(A)}$	No. iteration steps	Flops	
				Decomposition	Solve
Sparse LU		46.1	1	2.9×10^8	4.9×10^6
ILUSTAB	0.3	3.6	28	2.3×10^7	1.4×10^7
LUINC	0.1	4.9	16	4.1×10^7	7.7×10^6
	10^{-1}	1.0	–	6.4×10^5	–
	10^{-2}	3.7	–	1.0×10^7	–
	10^{-3}	7.4	29	3.2×10^7	2.0×10^7
	10^{-4}	12.3	9	6.6×10^7	7.1×10^6
	10^{-5}	17.0	9	1.0×10^8	5.0×10^6
ILUTP	10^{-1}	0.8	–		
	10^{-2}	1.4	–		
	10^{-3}	2.5	–		
	10^{-4}	3.5	–		
	10^{-5}	4.4	–		
AINVP	10^{-1}	127.6	–		
	10^{-2}	79.6	–		
	10^{-3}	65.0	29		

Here the direct solver produced significantly less fill-in for *nnc1374* (factor 14.6) than ILUSTAB.

- PORES: PORES1, PORES3 could be solved by ILUSTAB for $\tau = 0.3$ and ILUTP for $\tau = 0.1$. LUINC needed $\tau = 0.01$ for PORES3. The number of iteration steps was small except for PORES3, $\tau = 0.1$ and ILUTP which needed 248 steps, but for $\tau = 0.01$ the number of steps was small while the fill-in was still below the fill-in of the original matrix. For matrix PORES2, ILUSTAB needed a small number of iteration steps produced fill-in in the same order of the original system. LUINC produced less fill-in for $\tau = 10^{-2}$ but needed many more iteration steps. With $\tau = 10^{-3}$ the fill-in was comparable with that produced by LUINC and so was the number of iteration steps. ILUTP needed 10^{-3} but more fill-in while the number of iteration steps was comparable with those of the other two methods.
  Except *pores2* for which $\tau = 10^{-2}$ was sufficient, AINVP solved the other two matrices for $\tau = 0.1$. The fill-in for *pores1*, *pores3* and the number of iteration was small (less than a factor 2.7), while for *pores2* the number of iterations was small, but the fill-in was 8.9 more than for the initial matrix.
- SAYLOR: *saylr1/saylr3* were solved by ILUSTAB for $\tau = 0.3$ and ILUTP for $\tau = 0.1$. For SAYLR3, LUINC failed for all τ. For SAYLR4 ILUSTAB needed more fill-in than LUINC and ILUTP. But LUINC needed $\tau = 10^{-4}$ and ILUTP needed many more iteration steps. With decreasing $\tau(10^{-4}, 10^{-5})$ the results obtained by LUINC, ILUTP were in the same range as those computed by ILUSTAB. AINVP performed quite well on these matrices. *saylr1/saylr3* could be solved for $\tau = 0.1$ and even for *saylr4* $\tau = 0.01$ was sufficient. The number of iteration steps in these cases was small as well.
- SHERMAN: ILUSTAB solved all the matrices for $\tau = 0.3$, but for *sherman3* it needed 138 iteration steps. For the other matrics the iteration count was less than half as much. The fill-in was less than twice as much as the intial fill. The number of iterations was much lower for $\tau = 0.1$ but with more fill-in. *sherman2* could also be solved by ILUSTAB with $\tau = 0.3$ and less than half as many nonzeros as the initial matrix. In addition the number of iteration steps was at most 30. LUINC could only solve *sherman4*, *sherman5* for $\tau = 0.1$ and it needed 123 iteration steps for *sherman5*. For $\tau = 0.01$ it needed only a moderate number of iteration steps, but *sherman1* still could not be solved for $\tau = 10^{-2}$. *sherman2* could be solved for $\tau = 10^{-2}$ with results similar to those achieved by ILUSTAB. ILUTP could solve all matrices but for *sherman2* it needed $\tau = 10^{-5}$ but the fillin (factor 2.0) and the number of iteration steps was moderate for this choice of τ. For *sherman4* and $\tau = 0.1$ the number of iteration steps (449) was still big. This changed when using $\tau = 0.01$.
  Using $\tau = 0.01$ AINVP finally solved all matrices. For *sherman1*, *sherman4*, *sherman5* $\tau = 0.1$ was already sufficient. Especially *sherman2* did not cause any problems after $\tau = 0.01$ was chosen. The fill-in was only slightly more than for the initial matrix and GMRES(30) needed only 55 iteration steps.

The numerical examples have illustrated the robustness of taking the growth of the inverse triangular factors into account when computing an incomplete *LU* decomposition. Of course ILUSTAB is neither always the most efficient nor always the fastest (with respect to the flops) nor always the *ILU* with the smallest amount of fill-in. But in many cases it is a pretty good compromise between standard incomplete *LU* decompositions and the full sparse *LU* decomposition. In many examples it is not necessary to use a trial-and-error strategy for choosing the drop tolerance. The drop tolerance is automatically adapted with respect to the growth of the inverse factors. In several cases where a direct solver is competitive or superior to iterative methods (cf. Table 2) with respect to the number of flops, the fill-in for ILUSTAB is still moderate and often even less than that for LUINC, ILUTP. Conversely on some problems which cause trouble to direct solvers (cf. Table 3) ILUSTAB gains from its sparsity and being used as iterative solver.

The drawback of this algorithm is of course that it is more comparable with sparse direct solvers because it requires explicit knowledge of the Schur-complement. Clearly there are several problems where standard incomplete *LU* decompositions used as preconditioners give powerful iterative solvers. In these cases apparently ILUSTAB will be slower because one has a certain time consuming overhead for computing and administrating the approximate Schur-complement.

5. Conclusions

A version of an incomplete *LU* decomposition has been presented that performs dropping with respect to the growth of the inverses of the triangular factors. We have illustrated that the resulting preconditioner is very robust. Often one can avoid adapting the parameters to a specific matrix and still get a preconditioners that is computed in a sensible time with moderate fill-in. For many examples this has turned out to be a good compromise between sparse direct solvers and standard incomplete *LU* decompositions. Since this preconditioner shares several properties with sparse direct solvers, an implementation based on modified direct solvers seems to be reasonable. Currently codes from direct solvers like the Harwell–Subroutine–Library are under investigation to build this kind of preconditioner. Timing results using efficient implementations for bigger problems will be presented in a forthcoming paper.

References

[1] MATLAB—The language of technical computing. The MathWorks Inc., 1996.
[2] M. Benzi, J.K. Cullum, M. Tůma, Robust approximate inverse preconditioning for the conjugate gradient method, SIAM J. Sci. Comput. 22 (2000) 1318–1332.
[3] M. Benzi, C.D. Meyer, A direct projection method for sparse linear systems, SIAM J. Sci. Comput. 16 (5) (1995) 1159–1176.

[4] M. Benzi, C.D. Meyer, M. Tůma, A sparse approximate inverse preconditioner for the conjugate gradient method, SIAM J. Sci. Comput. 17 (1996) 1135–1149.
[5] M. Benzi, M. Tůma, A sparse approximate inverse preconditioner for nonsymmetric linear systems, SIAM J. Sci. Comput. 19 (3) (1998) 968–994.
[6] C.H. Bischof, Incremental condition estimation, SIAM J. Matrix Anal. Appl. 11 (2) (1990) 312–322.
[7] C.H. Bischof, J.G. Lewis, D.J. Pierce, Incremental condition estimation for sparse matrices, SIAM J. Matrix Anal. Appl. 11 (4) (1990) 644–659.
[8] M. Bollhöfer, Y. Saad, On the relations between ILUs and factored approximate inverses, Technical Report umsi-2001-67, Minnesota Supercomputer Institute, University of Minnesota, 2001, (submitted to SIAM Matrix Anal. Appl.).
[9] M. Bollhöfer, Y. Saad, A factored approximate inverse preconditioner with pivoting, SIAM Matrix Anal. Appl., to appear.
[10] A. Cline, C.B. Moler, G. Stewart, J. Wilkinson, An estimate for the condition number of a matrix, SIAM J. Numer. Anal. 16 (1979) 368–375.
[11] T.A. Davis, User's guide for the Unsymmetric-pattern MultiFrontal Package (UMFPACK), Techn. Report TR-95-004, Computer and Information Sciences Department, University of Florida, 1995.
[12] I. Duff, A. Erisman, J. Reid, Direct Methods for Sparse Matrices, Oxford University Press, Oxford, London, 1986.
[13] I. Duff, R. Grimes, J. Lewis, Sparse matrix test problems, ACM Trans. Math. Software 15 (1989) 1–14.
[14] I.S. Duff, R.G. Grimes, J.G. Lewis, Users' guide for the Harwell–Boeing sparse matrix collection (release 1), Technical Report RAL-TR-92-086, Rutherford Appleton Laboratory, Oxfordshire, England, 1992.
[15] I.S. Duff, J. Reid, The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations, ACM Trans. Math. Software 22 (1996) 187–226.
[16] R. Freund, G. Golub, N. Nachtigal, Iterative solution of linear systems, Acta Numer. (1992) 1–44.
[17] J.A. George, J.W. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.
[18] G. Golub, C.V. Loan, Matrix Computations, third ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[19] M. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Standards 49 (1952) 409–436.
[20] S. Kharchenko, L. Kolotilina, A. Nikishin, A. Yeremin, A reliable AINV-type preconditioning method for constructing sparse approximate inverse preconditioners in factored form, Technical report, Russian Academy of Sciences, Moscow, 1999.
[21] J. Meijerink, H.A.V. der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric *m*-matrix, Math. Comp. 31 (1977) 148–162.
[22] N. Munksgaard, Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method, ACM Trans. Math. Software 6 (1980) 206–219.
[23] National Institute of Standards. Matrix market. available online at http://math.nist.gov/MatrixMarket/.
[24] Y. Saad, ILUT: a dual treshold incomplete ILU factorization, Numer. Linear Algebra Appl. 1 (1994) 387–402.
[25] Y. Saad, SPARSKIT and sparse examples, NA Digest, 1994.
[26] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing, Boston, MA, 1996.
[27] Y. Saad, M. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 7 (1986) 856–869.
[28] M. Tismenetsky, A new preconditioning technique for solving large sparse linear systems, Linear Algebra Appl. 154–156 (1991) 331–353.